EXHIBIT 1

intel®

Willamette** Processor Software Developer's Guide

February, 2000

Copyright© 2000, Intel Corporation.
All Rights Reserved.

§Other product and corporate names may be trademarks of other companies and are used only for explanation and to the owner's benefit, without intent to infringe.

**Code Name for Future Intel IA-32 Processors

Order Number: 245355-001
World Wide Web: http://developer.intel.com/design/processor/index.htm

TABLE OF CONTENTS

PAGE

CHAPTER 1
INTRODUCTION TO THE WILLAMETTE PROCESSOR

1.1. THE WILLAMETTE PROCESSOR ARCHITECTURE 1-2
1.2. COMPATIBILITY WITH THE P6 FAMILY PROCESSOR ARCHITECTURE 1-3
1.3. THE WILLAMETTE PROCESSOR MICROARCHITECTURE 1-4
1.3.1. Higher Instruction Fetch Bandwidth 1-4
1.3.2. Increased Depth of Speculation 1-4
1.3.3. Integer And Floating Point Execution Core 1-5
1.3.4. Branch Prediction 1-5

CHAPTER 2
PROGRAMMING WITH THE STREAMING SIMD EXTENSIONS 2

2.1. STREAMING SIMD EXTENSIONS 2 FEATURE OVERVIEW 2-1
2.2. NEW DATA TYPES 2-2
2.3. STREAMING SIMD EXTENSIONS 2 REGISTERS 2-3
2.4. STREAMING SIMD EXTENSIONS 2 INSTRUCTIONS 2-3
2.4.1. Packed and Scalar Double-Precision Floating-Point Instructions 2-3
2.4.2. SIMD Integer Instruction Extensions 2-11
2.4.3. Cachcability Control and Memory Ordering Instructions 2-13
2.5. FLOATING-POINT TERMINOLOGY AND OPERATIONS 2-15
2.5.1. Real Numbers and Floating-Point Formats 2-16
2.5.2. Operating on NaNs 2-21
2.5.3. Streaming SIMD Extensions 2 Data Formats 2-22
2.5.4. MXCSR Register 2-25
2.5.5. Rounding Control Field 2-26
2.5.6. Flush to Zero 2-28
2.6. WRITING PROGRAMS USING THE STREAMING SIMD EXTENSIONS 2 2-28
2.6.1. Using the CPUID Instruction to Detecting the Existence of the Streaming SIMD Extensions 2 2-28
2.6.2. Updating Existing MMX Technology Routines Using 128-bit Integer Enhancements 2-30
2.6.3. Interaction of Streaming SIMD Extensions 2 and x87-FPU and MMX Instructions 2-31
2.6.4. Interaction of Packed or Scalar Floating-Point Instructions with the x87 FPU and MMX Instructions 2-31
2.6.5. Caller-Save Requirement for Function Calls 2-33
2.6.6. Cachcability Hint Instructions 2-33
2.6.7. Branching on Streaming SIMD Extensions 2 Arithmetic Operations 2-34
2.6.8. Saving the Streaming SIMD Extensions and Streaming SIMD Extensions 2 State 2-35
2.6.9. Initialization of Streaming SIMD Extensions and Streaming SIMD Extensions 2 Technology 2-35
2.6.10. Interfacing with Streaming SIMD Extensions 2 Procedures and Functions 2-36
2.7. HANDLING EXCEPTIONS IN STREAMING SIMD EXTENSIONS 2 OPERATIONS 2-36
2.7.1. Non-Numeric Exceptions 2-37
2.7.2. SIMD Floating-Point Exceptions 2-38
2.7.3. Streaming SIMD Extensions 2 Floating-Point Exception Conditions 2-42

TABLE OF CONTENTS

PAGE

2.8. SYSTEM PROGRAMMING MODEL 2-46
2.8.1. Enabling Streaming SIMD Extensions 2 Support. 2-46
2.8.2. Device Not Available Exception. 2-47
2.8.3. Streaming SIMD Extensions 2 Emulation 2-47
2.8.4. Numeric Error flag and IGNNE# 2-47

CHAPTER 3
STREAMING SIMD EXTENSIONS 2 INSTRUCTION SET
3.1. NOTATION. 3-1
3.2. PACKED AND SCALAR DOUBLE-PRECISION FLOATING-POINT INSTRUCTIONS 3-2
ADDPD—Packed Double-Precision Floating-Point Add3-3
ADDSD—Scalar Double-Precision Floating-Point Add3-5
ANDNPD—Bitwise Logical AND NOT for Double-Precision Floating-Point3-7
ANDPD—Bitwise Logical AND for Double-Precision Floating-Point3-9
CMPPD—Packed Double-Precision Floating-Point Compare3-11
CMPSD—Scalar Double-Precision Floating-Point Compare3-15
COMISD—Scalar Ordered Double-Precision Floating-Point Compare and Set EFLAGS 3-19
CVTDQ2PD—Packed Doubleword Signed Integer to Packed Double-Precision Floating-Point Conversion 3-21
CVTPD2PI—Packed Double-Precision Floating-Point to Packed Doubleword Integer Conversion 3-23
CVTPD2DQ—Packed Double-Precision Floating-Point to Packed Doubleword Integer Conversion. 3-26
CVTPD2PS—Packed Double-Precision Floating-Point to Packed Single-Precision Floating-Point Conversion 3-28
CVTPI2PD—Packed Signed Doubleword Integer to Packed Double-Precision Floating-Point Conversion 3-30
CVTPS2PD—Packed Single-Precision to Packed Double-Precision Floating-Point Conversion 3-33
CVTSD2SI—Scalar Double-Precision Floating-Point to Signed Doubleword Integer Conversion 3-35
CVTSD2SS—Scalar Double-Precision Floating-Point to Scalar Single-Precision Floating-Point Conversion 3-37
CVTSI2SD—Scalar signed INT32 to Double-Precision Floating-Point Conversion 3-39
CVTSS2SD—Scalar Single-Precision Floating-Point to Scalar Double-Precision Floating-Point Conversion. 3-41
DIVPD—Packed Double-Precision Floating-Point Divide. 3-50
DIVSD—Scalar Double-Precision Floating-Point Divide 3-52
MAXPD—Packed Double-Precision Floating-Point Maximum. 3-54
MAXSD—Scalar Double-Precision Floating-Point Maximum 3-56
MINPD—Packed Double-Precision Floating-Point Minimum. 3-58
MINSD—Scalar Double-Precision Floating-Point Minimum 3-60
MOVAPD—Move Aligned Two Packed Double-Precision Floating-Point. 3-62
MOVHPD—Move High Packed Double-Precision Floating-Point 3-65
MOVLPD—Move Low Packed Double-Precision Floating-Point. 3-67

PAGE

MOVMSKPD—Move Mask To Integer 3-69
MOVSD—Move Scalar Double-Precision Floating-Point 3-71
MOVUPD Move Unaligned Two Packed Double-Precision Floating-Poin. 3-73
MULPD Packed Double-Precision Floating-Point Multiply 3-76
MULSD -Scalar Double-Precision Floating-Point Multiply 3-78
ORPD—Bitwise Logical OR for Double-Precision Floating-Point Data3-80
SHUFPD—Shuffle Double-Precision Floating-Point 3-82
SQRTPD—Packed Double-Precision Floating-Point Square Root. 3-84
SQRTSD Scalar Double-Precision Floating-Point Square Root. 3-86
SUBPD -Packed Double-Precision Floating-Point Subtrac. 3-88
SUBSD—Scalar Double-Precision Floating-Point Subtract 3-90
UCOMISD—Unordered Scalar Double-Precision Floating-Point Compare
and Set EFLAGS. 3-92
UNPCKHPD—Unpack High Packed Double-Precision Floating-Point Data. 3-94
UNPCKLPD—Unpack Low Packed Double-Precision Floating-Point Data 3-96
XORPD—Bitwise Logical XOR for Double-Precision Floating-Point Data 3-98

3.3. SIMD INTEGER INSTRUCTIONS 3-100
CVTDQ2PS: Packed Signed Doubleword Integer to Packed
Single-Precision Floating-Point Conversion. 3-101
CVTPS2DQ—Packed Single-Precision Floating-Point to Packed Doubleword
Integer Conversion. 3-103
CVTTPS2DQ—Packed Single-Precision Floating-Point to Packed Signed
Doubleword Integer Conversion (Truncate). 3-105
MOVD - Move Doubleword 3-107
MOVDQA -Move Aligned Double Quadword 3-109
MOVDQU—Move Unaligned Double Quadword 3-111
MOVDQ2Q—Move Quadword. 3-113
MOVQ2DQ—Move Quadword. 3-115
MOVQ—Move Quadword. 3-117
PACKSSWB/PACKSSDW - Pack with Signed Saturation 3-119
PACKUSWB—Pack with Unsigned Saturation 3-121
PADDB/PADDW/PADDD—Packed Add 3-123
PADDQ—Packed Add Quadwords 3-126
PADDQ—Packed Add Quadword 128 Bits 3-128
PADDSB/PADDSW -Packed Add with Saturation. 3-130
PADDUSB/PADDUSW-Packed Add Unsigned with Saturation 3-132
PAND—Bitwise Logical AND 3-134
PANDN—Bitwise Logical And Not 3-136
PAVGB/PAVGW—Packed Average. 3-138
PCMPEQB/PCMPEQW/PCMPEQD Packed Compare for Equal. 3-141
PCMPGTB/PCMPGTW/PCMPGTD—Packed Compare for Greater Than 3-143
PEXTRW—Extract Word 3-145
PINSRW- Insert Word 3-146
PMADDWD- Packed Multiply and Add 3-148
PMAXSW—Packed Signed Word Integer Maximum. 3-150
PMAXUB- Packed Unsigned Byte Integer Maximum 3-152
PMINSW Packed Signed Integer Word Minimum. 3-154

TABLE OF CONTENTS

PAGE

PMINUB - -Packed Unsigned Byte Integer Minimum 3-156
PMOVMSKB—Move Byte Mask To Integer 3-158
PMULHW—Packed Multiply High 3-160
PMULHUW—Packed Multiply High Unsigned 3-162
PMULLW - Packed Multiply Low 3-164
PMULUDQ - -Multiply Doubleword Unsigned 3-166
PMULUDQ—Packed Multiply Doubleword Unsigned 3-168
POR—Bitwise Logical OR 3-170
PSADBW—Packed Sum of Absolute Differences 3-172
PSHUFD—Packed Shuffle Doubleword 3-174
PSHUFHW -Packed Shuffle High Words 3-176
PSHUFLW—Packed Shuffle Low Word 3-178
PSLLDQ ·Packed Shift Left Logical Double Quadword 3-180
PSLLW/PSLLD/PSLLQ — Packed Shift Left Logical 3-181
PSRAW/PSRAD—Packed Shift Right Arithmetic 3-184
PSRLDQ—Packed Shift Right Logical Double Quadword 3-186
PSRLW/PSRLD/PSRLQ— Packed Shift Right Logica 3-188
PSUBB/PSUBW/PSUBD- Packed Subtract 3-191
PSUBQ— Packed Subtract Quadword 3-194
PSUBQ—Packed Subtract Quadword 3-196
PSUBSB/PSUBSW —Packed Subtract with Saturation 3-198
PSUBUSB/PSUBUSW—Packed Subtract Unsigned with Saturation 3-201
PUNPCKH—Unpack High Packed Data 3-203
PUNPCKL—Unpack Low Packed Data 3-206
PXOR—Bitwise Logical XOR 3-209

3.4. CACHEABILITY CONTROL AND MEMORY ORDERING INSTRUCTIONS 3-211
CLFLUSH—Cache Line Flush 3-212
LFENCE—Load Fence 3-214
MASKMOVDQU—Byte Mask Write Unaligned 3-215
MFENCE—Memory Fence 3-217
MOVNTPD· ·Move Aligned Four Packed Double-Precision Floating-Point Non-Temporal 3-218
MOVNTDQ—Move Double Quadword Non-Temporal 3-220
MOVNTI -Move Integer Non-Temporal 3-222
PAUSE- REP NOP 3-224

3.5. MODIFIED INSTRUCTIONS 3-225
3.5.1. CPUID Instruction 3-225
3.5.2. RDPMC Instruction 3-228
3.5.3. Branch Hints 3-228

CHAPTER 4
CODE OPTIMIZATION

4.1. CODE OPTIMIZATION GUIDELINES 4-1
4.1.1. Improve Branch Predictability 4-1
4.1.2. Scheduling 4-2
4.1.3. Code Selection 4-2
4.1.4. Memory 4-3
4.1.5. General Principles for Code and Data Layout 4-3

TABLE OF CONTENTS

PAGE

4.1.6. Make Use of Prefetching 4-4
4.1.7. New Instructions 4-4
4.1.8. Code Size4-4
4.2. NOTABLE DIFFERENCES BETWEEN THE WILLAMETTE AND P6 FAMILY PROCESSORS 4-5
4.2.1. Code Selection 4-5
4.2.2. New Instructions 4-5

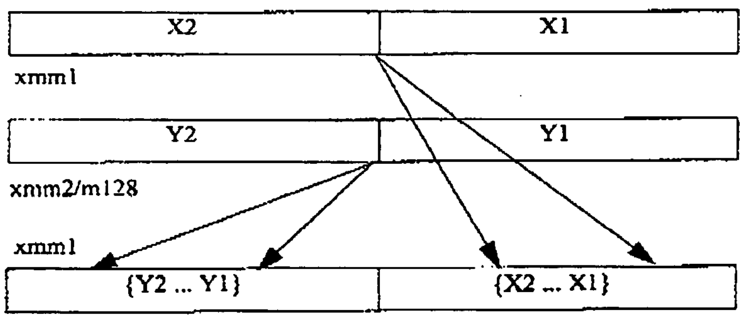
APPENDIX A
STREAMING SIMD EXTENSIONS 2 INSTRUCTION SUMMARY

SHUFPD—Shuffle Double-Precision Floating-Point

Instruction	Description
SHUFPD xmm1, xmm2/m128, imm8	Shuffle packed double-precision floating-point numbers.

Description

Shuffles either of the two packed double-precision floating-point numbers from xmm1 to the low quadword of xmm1; shuffles either of the two packed double-precision floating-point numbers from xmm2/m128 to the high quadword of xmm1. Bit 0 of the immediate field selects which of the two input double-precision floating-point numbers will be put in the low quadword of the result; bit 1 selects which of the two input double-precision floating-point numbers will be put in the high quadword of the result.



Operation

```
fp_select = (imm8 >> 0) & 0x1;
xmm1[63-0] =        (fp_select == 0) ? xmm1[63-0]  :
                    (fp_select == 1) ? xmm1[127-64];
fp_select = (imm8 >> 1) & 0x1;
xmm1[127-64] =      (fp_select == 0) ? xmm2/m128[63-0]  :
                    (fp_select == 1) ? xmm2/m128[127-64];
```

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.

#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions 2 instructions numeric exception (CR4.OSXMMEXCPT =1).
#UD	For an unmasked Streaming SIMD Extensions 2 instructions numeric exception (CR4.OSXMMEXCPT =0).
	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.WNI(EDX bit 26) = 0.

Real-Address Mode Exceptions

#GP(0)	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
Interrupt 13	If any part of the operand lies outside the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#XM	For an unmasked Streaming SIMD Extensions 2 instructions numeric exception (CR4.OSXMMEXCPT =1).
#UD	For an unmasked Streaming SIMD Extensions 2 instructions numeric exception (CR4.OSXMMEXCPT =0).
	If CR0.EM = 1.
	If CR4.OSFXSR (bit 9) = 0.
	If CPUID.WN(EDX bit 26) = 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault.

Numeric Exceptions

None.

PSHUFD—Packed Shuffle Doubleword

Instruction	Description
PSHUFD xmm1, xmm2/m128, imm8	Shuffle the doublewords in xmm2/mem128 based on the encoding in imm8 and store result in xmm1.

Description

Shuffles the four doublewords in xmm2/mem128 in the order selected by imm8 and stores the result in xmm1. Bits 1 and 0 of imm8 encode the source for destination doubleword 0 (xmm1[31-0]), bits 3 and 2 encode for doubleword 1, bits 5 and 4 encode for doubleword 2, and bits 7 and 6 encode for doubleword 3 (xmm1[127-96]). Similarly, the two bit encoding represents which source doubleword is to be used, e.g., an binary encoding of 10 indicates that source doubleword 2 (xmm2/mem128[95-64]) will be used.

Operation

```
xmm1[31-0]   = (xmm2/m128 >> (imm8[1-0] * 32) )[31-0]
xmm1[63-32]  = (xmm2/m128 >> (imm8[3-2] * 32) )[31-0]
xmm1[95-64]  = (xmm2/m128 >> (imm8[5-4] * 32) )[31-0]
xmm1[127-96] = (xmm2/m128 >> (imm8[7-6] * 32) )[31-0]
```

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.WNI(EDX bit 26) = 0.

Real-Address Mode Exceptions

#GP(0)	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
Interrupt 13	If any part of the operand lies outside the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.

#UD	If CR0.EM = 1.
	If CR4.OSFXSR (bit 9) = 0.
	If CPUID.WN(EDX bit 26) = 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault.

Numeric Exceptions

None.

PSHUFHW—Packed Shuffle High Words

Instruction	Description
PSHUFHW xmm1, xmm2/m128, imm8	Shuffle the high words in xmm2/mem128 based on the encoding in imm8 and store the result in xmm1.

Description

Shuffles the four high words in xmm2/mem128 in the order selected by imm8 and stores the result in the high quadword of xmm1. Bits 1 and 0 of imm8 encode the source for destination word 4 (xmm1[79-64]), bits 3 and 2 encode for word 5, bits 5 and 4 encode for word 6, and bits 7 and 6 encode for word 7 (xmm1[127-112]). Similarly, the two bit encoding represents which source word is to be used, e.g., a binary encoding of 10 indicates that source word 6 (XMM2[111-96] or Mem[111-96]) will be used. The low quadword of the destination register is written with the low 64 bits of the source register.

Operation

```
if (source == m128) {
    xmm1[79-64]   = (m128 >> (imm8[1-0] * 16) )[79-64]
    xmm1[95-80]   = (m128 >> (imm8[3-2] * 16) )[79-64]
    xmm1[111-96]  = (m128 >> (imm8[5-4] * 16) )[79-64]
    xmm1[127-112] = (m128 >> (imm8[7-6] * 16) )[79-64]

    xmm1[63-0] = m128[63-0];
} else {
    xmm1[79-64]   = (xmm2 >> (imm8[1-0] * 16) )[79-64]
    xmm1[95-80]   = (xmm2 >> (imm8[3-2] * 16) )[79-64]
    xmm1[111-96]  = (xmm2 >> (imm8[5-4] * 16) )[79-64]
    xmm1[127-112] = (xmm2 >> (imm8[7-6] * 16) )[79-64]

    xmm1[63-0] = xmm2[63-0];
}
```

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.

	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.WNI(EDX bit 26) = 0.

Real-Address Mode Exceptions

#GP(0)	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
Interrupt 13	If any part of the operand lies outside the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR (bit 9) = 0.
	If CPUID.WN(EDX bit 26) = 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault.

Numeric Exceptions

None.

PSHUFLW—Packed Shuffle Low Word

Instruction	Description
PSHUFLW xmm1, xmm2/m128, imm8	Shuffle the low words in xmm2/mem128 based on the encoding in imm8 and store in xmm1.

Description

Shuffles the four low words in xmm2/mem128 in the order selected by imm8 and stores the result in the low quadword of xmm1.Bits 1 and 0 of imm8 encode the source for destination word 0 (xmm1[15-0]), bits 3 and 2 encode for word 1, bits 5 and 4 encode for word 2, and bits 7 and 6 encode for word 3 (xmm1[63-48]). Similarly, the two bit encoding represents which source word is to be used, e.g., an binary encoding of 10 indicates that source word 2 (xmm2/mem128[47-32]) will be used. The high quadword of the destination register is written with the high 64 bits of the source register.

Operation

```
if (source == m128) {
    xmm1[15-0]  = (m128 >> (imm8[1-0] * 16) )[15-0]
    xmm1[31-16] = (m128 >> (imm8[3-2] * 16) )[15-0]
    xmm1[47-32] = (m128 >> (imm8[5-4] * 16) )[15-0]
    xmm1[63-48] = (m128 >> (imm8[7-6] * 16) )[15-0]

    xmm1[127-64] = m128[127-64];
} else {
    xmm1[15-0]  = (xmm2 >> (imm8[1-0] * 16) )[15-0]
    xmm1[31-16] = (xmm2 >> (imm8[3-2] * 16) )[15-0]
    xmm1[47-32] = (xmm2 >> (imm8[5-4] * 16) )[15-0]
    xmm1[63-48] = (xmm2 >> (imm8[7-6] * 16) )[15-0]

    xmm1[127-64] = xmm2[127-64];
}
```

Protected Mode Exceptions

#GP(0)	For an illegal memory operand effective address in the CS, DS, ES, FS or GS segments.
	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
#SS(0)	For an illegal address in the SS segment.
#PF(fault-code)	For a page fault.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.

	If CR4.OSFXSR(bit 9) = 0.
	If CPUID.WNI(EDX bit 26) = 0.

Real-Address Mode Exceptions

#GP(0)	If memory operand is not aligned on a 16-byte boundary, regardless of segment.
Interrupt 13	If any part of the operand lies outside the effective address space from 0 to 0FFFFH.
#NM	If TS bit in CR0 is set.
#UD	If CR0.EM = 1.
	If CR4.OSFXSR (bit 9) = 0.
	If CPUID.WN(EDX bit 26) = 0.

Virtual-8086 Mode Exceptions

Same exceptions as in Real Address Mode

#PF(fault-code)	For a page fault.

Numeric Exceptions

None.